From First Principles: Thinking About Prompt Engineering

How Context Shapes LLM Behavior—and How to Make Its Outputs More Reliable

Elleira

From First Principles: Thinking About Prompt Engineering

How Context Shapes LLM Behavior—and How to Make Its Outputs More Reliable

Elleira

This book is available at
https://leanpub.com/promptengineeringfromfirstprinciples

This version was published on 2026-07-18

Contents

Why Do We Need Prompt Engineering?

Ask an LLM the same question in two different ways, and you may get two very different answers. Give it an example, and the answer often becomes more consistent. Add the right context, and the model may suddenly produce the response you wanted all along.

Nothing about the model itself has changed, so why can a few lines of text change its behavior so much?

That question is the beginning of prompt engineering. To answer it, we first need to look beneath the chat interface and trace the machinery that an LLM inherited.

1. The Lineage Behind the Chat Interface

The conversational interface makes an LLM feel unlike earlier software. You type a question, and something on the other side appears to understand what you mean, reason about it, and compose an answer. Yet the novelty of this experience can hide the machinery underneath. An LLM is not disconnected from the history of machine learning; it is the latest development in that history:

Artificial intelligence -> machine learning -> neural networks -> deep learning -> large language models.

Each step in this lineage emerged because the previous approach ran into a limit. Imagine that we want software to recognize spam. We could write rules for suspicious phrases, senders, and links, but new forms of spam would appear faster than we could maintain the rulebook. A more adaptable approach is to show the software many previous messages and let it discover which patterns tend to separate spam from legitimate email. This shift—from programming every rule to learning patterns from data—is the basic move behind **machine learning**.

Whether the rules are written by a programmer or learned from data, the task has the same outer shape: an email goes in, and a classification comes out. Machine learning changes how that relationship is discovered, not the fact that the result depends on what the system receives.

Once we ask machines to work with images, audio, or language, however, the path from what they receive to what they must produce becomes far more complicated. A sentence cannot be classified from one word or one fixed rule; its meaning emerges from many features interacting at once. **Neural networks** help because they can learn these layered relationships. What began with relatively simple structures such as the perceptron developed into multilayer networks capable of representing increasingly complex transformations.

As researchers added more layers, larger datasets, and more computation, these networks became the foundation of **deep learning**. An image could become a label, audio could become a transcript, and one piece of text could become another. Large language models apply this machinery to language at extraordinary scale. The inputs and outputs have changed, and the transformation between them has become enormously more capable, but the dependency remains: what the model produces still depends on what it receives.

That recurring dependency gives us the first mental model we need:

> However intelligent the output appears, the model is still producing it from an input.

We have described that relationship informally. Now we can express it more precisely.

2. The Input-Output Relationship

Suppose we give a model an image and ask whether it contains a cat. The model receives the image, processes its features, and produces a prediction. We can express this relationship with a simple abstraction:

$$\text{Model}(x) = y$$

Here:

- x is the input, such as an image, a row of data, or a piece of text.
- Model is the learned function that transforms the input.
- y is the resulting prediction or output.

The purpose of training is to make this mapping useful. We begin with a large collection of example inputs paired with desired outputs, (x, y). The model makes a prediction for each input, compares that prediction with the expected output, and adjusts its internal parameters to reduce the difference. Repeating this process across many examples gradually shapes the model into an approximation of the relationship contained in the data.

Neural networks became especially important because they can approximate extremely complex functions while still being trainable through gradient-based optimization. As the networks, datasets, and available computation grew, the mappings they could learn became increasingly powerful.

An LLM uses this machinery to learn patterns in language. Given the text in its input, it assigns probabilities to possible next tokens and generates a continuation from those possibilities. We will examine that process in more detail later. For now, we can preserve the same simple relationship:

$$\text{LLM}(x) = y$$

The simplicity of this equation hides an important consequence:

> An LLM does not respond directly to your intention. It responds to the input through which your intention is expressed.

That difference between intention and input is where prompt engineering begins. But before we can improve the input, we need to know what it includes.

3. The Part We Can Control

For an LLM, x is not merely the final question you type. It can include the instructions that define the task, background knowledge that frames the subject, examples that demonstrate the desired pattern, constraints that limit the answer, documents retrieved from another system, and the conversation

that came before. The model generates its response from this entire context, not from your unspoken goal.

This distinction matters because most of us cannot change the model itself. Training or fine-tuning a large model requires data, computation, expertise, and money, while the models we use through products and APIs usually arrive with their parameters already fixed. We cannot easily change the function, but we can change what we give it. The practical variable available to us is x.

Prompt engineering is the deliberate design of x to produce a more useful y.

This does not mean every weak answer is caused by a weak prompt. Models have limits, generation contains uncertainty, and some tasks exceed what a particular model can do. Yet the quality of the input still matters within those limits. A vague prompt leaves the model to infer the task, the relevant context, and the desired form of the answer. A contradictory prompt pulls it in competing directions. A well-designed prompt reduces these ambiguities and makes a useful result more likely.

Prompt engineering is therefore not a collection of magic phrases. It is the practical work of translating an intention into an input the model can act on: choosing the instructions it should follow, the context it should use, the examples it should imitate, and the boundaries it should respect.

That is the first principle of this discipline:

> If x is the part of the system we can control, then improving x is the most direct way to improve y. Prompt engineering uses structure, context, and constraints to reduce ambiguity in x, narrowing the range of plausible outputs toward the y we intend.

That principle gives us a destination. The rest of this book is about how to design x deliberately rather than by trial and error.

Zero-Shot and Few-Shot Prompting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/promptengineeringfromfirstprinciples.

Zero-Shot Prompting: Guidance Without Demonstrations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/promptengineeringfromfirstprinciples.

The Anatomy of a Zero-Shot Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/promptengineeringfromfirstprinciples.

Few-Shot Prompting: Guidance Through Demonstrations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/promptengineeringfromfirstprinciples.

Demonstrations Reveal Hidden Boundaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/promptengineeringfromfirstprinciples.

Why Can One Example Teach the Wrong Lesson?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/promptengineeringfromfirstprinciples.

Does Adding More Shots Always Improve the Result?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/promptengineeringfromfirstprinciples.

Building a Small but Powerful Example Set

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/promptengineeringfromfirstprinciples.